Lie group proposal : a (very) quick overview

Mathieu Gautier
CEA – List, Laboratoire de Simulation Interactive

Short introduction

- A lie group is a :
  - Group
  - Differentiable manifold
    - Used in differential calculus
    - Have a tangent space on each point : Lie Algebra
- Commonly used group :
  - Rotation matrices (2d : SO(2), 3d : SO(3))
  - Orthogonal group : group of orthogonal matrices
  - Euclidian group : isometries
  - Other : quantum physics, particle physics

Short introduction

- Interesting properties
  - A Lie algebra is a vector space
  - General Linear Group (group of invertible matrices)
    - The exponential map is map from the Lie Algebra to the Lie Group
    - For a Matrix A :

$$\exp(A) = 1 + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots$$

- More on : http://en.wikipedia.org/wiki/Lie_group

Lie group module requirements

- General linear group
  - Based on matrices

- Algorithms could be generalized (rotation matrices, orthogonal matrices, etc.)
  - Defines the following method
    - Exp, log and derivatives
    - Adjoint and coadjoint
    - Bracket for the Lie Algebra
  - Interpolation, statistic analysis, etc.

Lie Group : previously in Eigen

- Rotation : through Quaternion, etc.
- First proposal : limited to SO(3) and SE(3) : used for rigid bodies simulations
  - Introduce several classes similar to the Quaternion class
- Second proposal (Maxime Tournier) : a more general approach
  - A templated class described the Lie group structure for a type T
  - Static methods implement composition, exp, log, etc.

Lie Group : previously in Eigen

```
template<class G>
struct Lie {
  typedef some_type algebra;
  typedef some_type coalgebra;

  static G id();
  static G inv(const G&);
  static G comp(const G&, const G& );

  // default-constructible
  typedef some_functor exp;
  typedef some_functor log;

  // G-constructible
  typedef some_functor ad;
  typedef some_functor ad_T;
};
```

- New groups could be expressed from existing ones

```
template<class G1, class G2>
struct Lie< std::pair<G1, G2> > {

  // ...

};
```

Improving the design

- The first proposal is too limited

- The second proposal is more generic but more tedious to write.

  g2 = Lie<G>::comp(g, g)

- Solution : wrapper

adding a reference

```
template<class G>
struct Lie {
    ...

    G& element;
};
```

adding the object

```
template<class G>
struct Lie {
    ...

    G element;
};
```

The wrapper solution

- Comparison of both wrappers

Reference
Dev : use wrapper and base class
User : use only base class

```
template<G>
G fast_exp(const G& _g, int n)
{
  LieWrapper<G> g(_g);

  if( n == 0 )
    return Lie<G>::Identity();

  if( n % 2 )
    return g * fast_exp(g*g, n/2);
  else
    return fast_exp( g*g, n/2);
}
```

Instance
Dev : Use only wrapper
User : Use only wrapper

```
template<class G> LieGroup<G>
fastExp(const LieGroup<G>& g, int n){
  if(n==0)
    return LieGroup<G>::Identity();

  if(n%2)
    return g*fastExp(g*g,n/2);
  else
    return fastExp(g*g,n/2);
}
typedef LieGroup<Quaterniond> Rotation;
```

Remarks and Conclusion

- Could not benefits from expression template, but
  - Operations maybe complex : not much improvement
  - Objects are smalls in our cases (maxime's and mine)
    - NRVO optimization can be enough
- Integration with the geometry module
  - How do we link LieGroup<Quaternion>, LieGroup<Matrix<Scalar, 3,3> > and Eigen::RotationBase and Transform ?
- Is this design good enough to add new objects to the Eigen library ?